# Data Cleaning with R

NCSU LIBRARIES

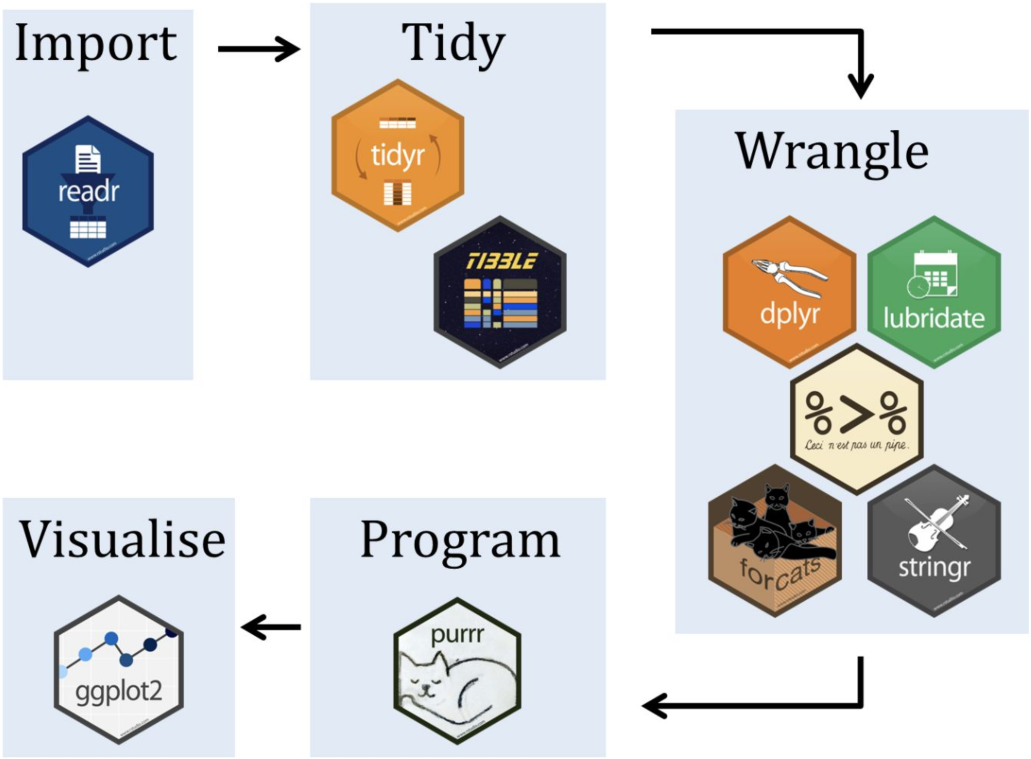
# Messy data can be caused by

- Error in data entry
- Incomplete data entry
- Bringing data in from multiple sources or streams
- Changes in how the data is organized over time
- Problems introduced when data is exported/shared/opened

# What data cleaning tasks are common?

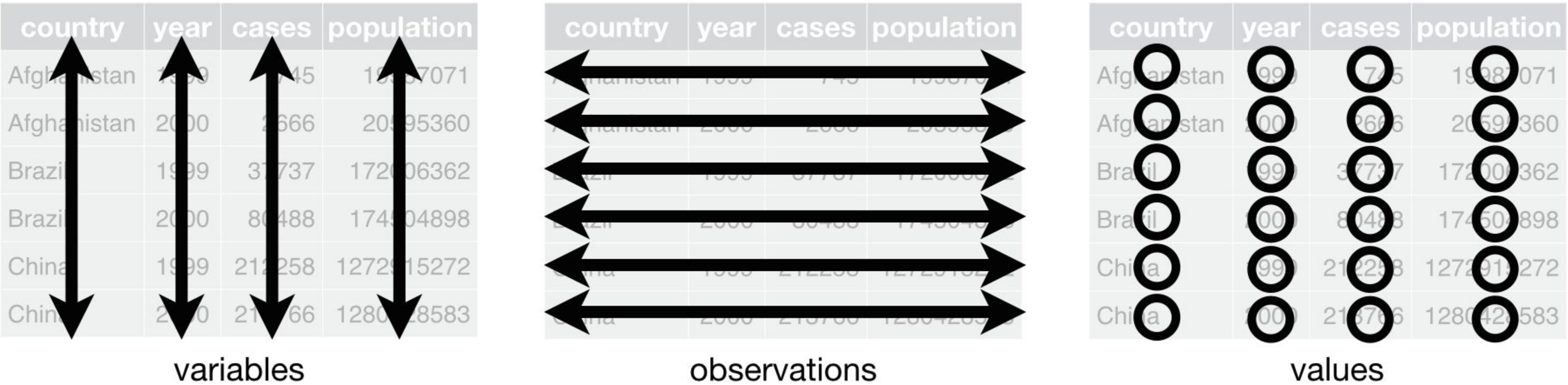
- Fix errors in the data
- Removing duplicate records
- Removing outliers
- Stripping whitespace, removing unwanted characters, changing case
- Restructuring the data
- Splitting data into multiple columns
- Merging data

# tidyverse: R packages for data cleaning and visualization



# tidyverse packages expect *tidy* data

(packages tidyr and tibble are designed to help you get the data in tidy format)

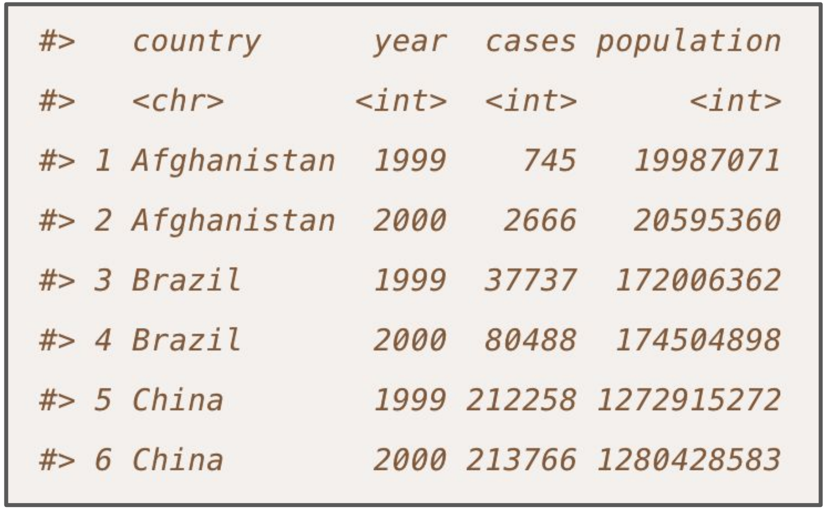


“There are three interrelated rules which make a dataset tidy:

1. Each **variable** must have its own column.
2. Each **observation** must have its own row.
3. Each **value** must have its own cell.”

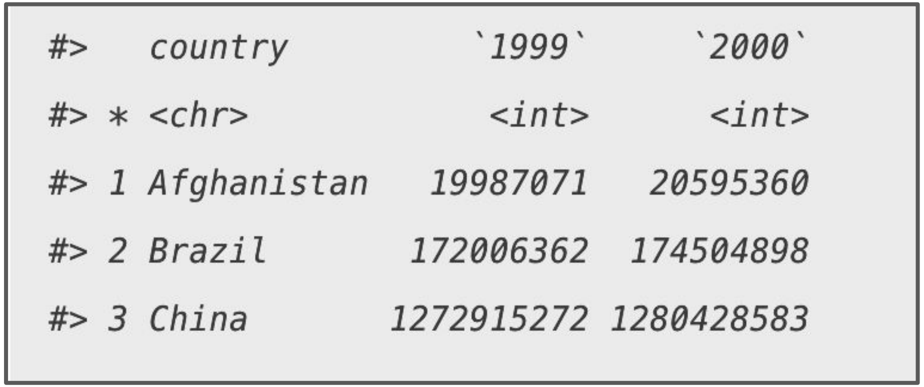
-Hadley Wickham, *R for Data Science*

Tidy data



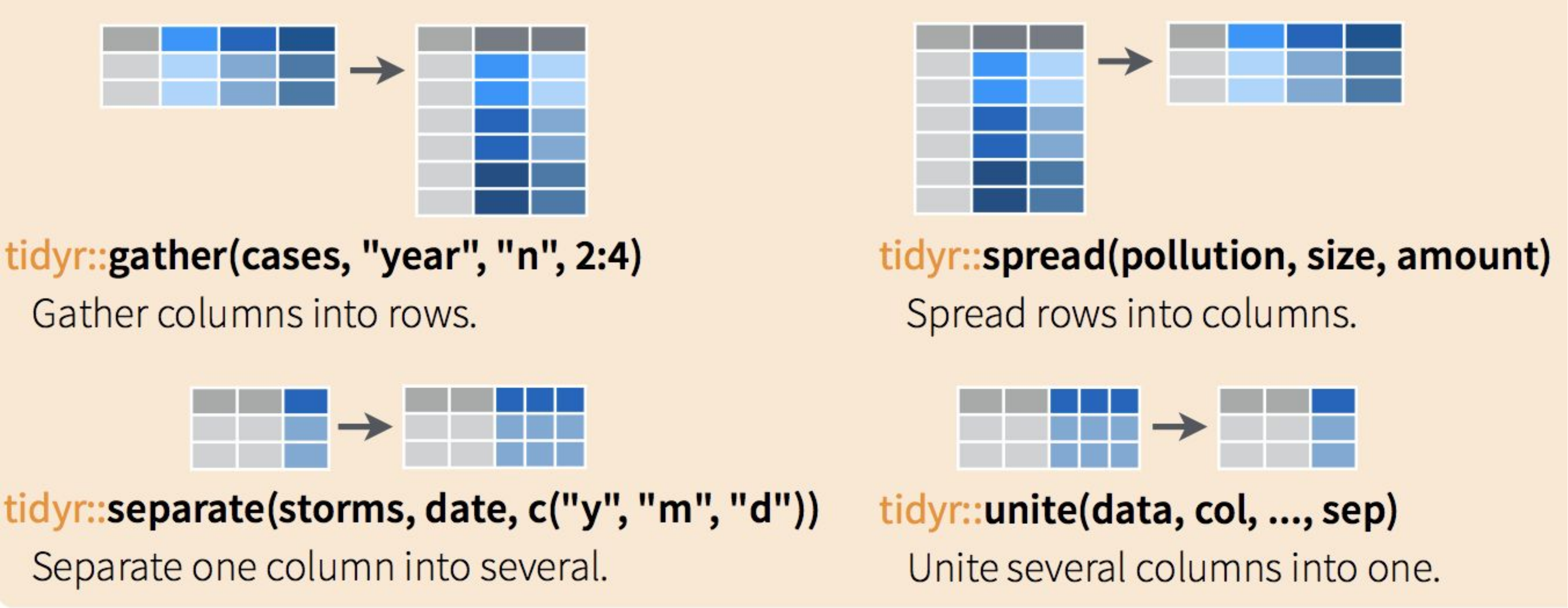
```
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

NOT tidy data



```
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

# **tidyr** package helps with restructuring data



`tidyr::gather(cases, "year", "n", 2:4)`

Gather columns into rows.

`tidyr::spread(pollution, size, amount)`

Spread rows into columns.

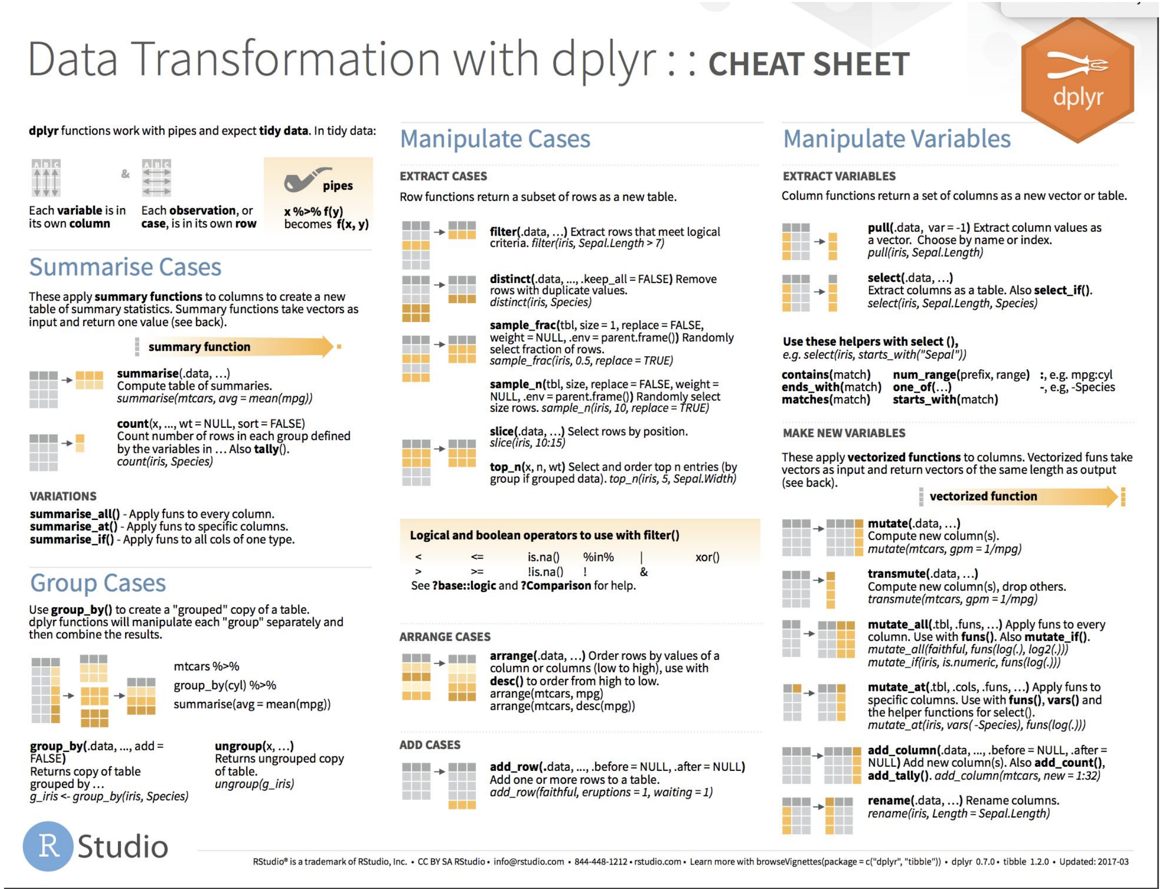
`tidyr::separate(storms, date, c("y", "m", "d"))`

Separate one column into several.

`tidyr::unite(data, col, ..., sep)`

Unite several columns into one.

# **dplyr** package helps with data transformation



## Data Transformation with dplyr : : CHEAT SHEET

dplyr

**dplyr** functions work with pipes and expect **tidy data**. In tidy data:

A B C
&
A B C
pipes

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

**x %>% f(y)** becomes **f(x, y)**

### Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

**summarise**(.data, …)
Compute table of summaries.
*summarise(mtcars, avg = mean(mpg))*

**count**(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
*count(iris, Species)*

**VARIATIONS**

**summarise_all()** - Apply funs to every column.
**summarise_at()** - Apply funs to specific columns.
**summarise_if()** - Apply funs to all cols of one type.

### Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

**group_by**(.data, ..., add = FALSE)
Returns copy of table grouped by …
*g_iris <- group_by(iris, Species)*

**ungroup**(x, …)
Returns ungrouped copy of table.
*ungroup(g_iris)*

### Manipulate Cases

**EXTRACT CASES**

Row functions return a subset of rows as a new table.

**filter**(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

**distinct**(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
*distinct(iris, Species)*

**sample_frac**(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
*sample_frac(iris, 0.5, replace = TRUE)*

**sample_n**(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

**slice**(.data, …) Select rows by position.
*slice(iris, 10:15)*

**top_n**(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

**Logical and boolean operators to use with filter()**

| | | | | | |
|---|---|---|---|---|---|
| < | <= | is.na() | %in% | \| | xor() |
| > | >= | !is.na() | ! | & | |

See **?base::logic** and **?Comparison** for help.

**ARRANGE CASES**

**arrange**(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

**ADD CASES**

**add_row**(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
*add_row(faithful, eruptions = 1, waiting = 1)*

### Manipulate Variables

**EXTRACT VARIABLES**

Column functions return a set of columns as a new vector or table.

**pull**(.data, var = -1) Extract column values as a vector. Choose by name or index.
*pull(iris, Sepal.Length)*

**select**(.data, …)
Extract columns as a table. Also **select_if()**.
*select(iris, Sepal.Length, Species)*

**Use these helpers with select (),**
*e.g. select(iris, starts_with("Sepal"))*

| | | |
|---|---|---|
| **contains**(match) | **num_range**(prefix, range) | **:**, e.g. mpg:cyl |
| **ends_with**(match) | **one_of**(…) | **-**, e.g, -Species |
| **matches**(match) | **starts_with**(match) | |

**MAKE NEW VARIABLES**

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

**mutate**(.data, …)
Compute new column(s).
*mutate(mtcars, gpm = 1/mpg)*

**transmute**(.data, …)
Compute new column(s), drop others.
*transmute(mtcars, gpm = 1/mpg)*

**mutate_all**(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
*mutate_all(faithful, funs(log(.), log2(.)))*
*mutate_if(iris, is.numeric, funs(log(.)))*

**mutate_at**(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
*mutate_at(iris, vars( -Species), funs(log(.)))*

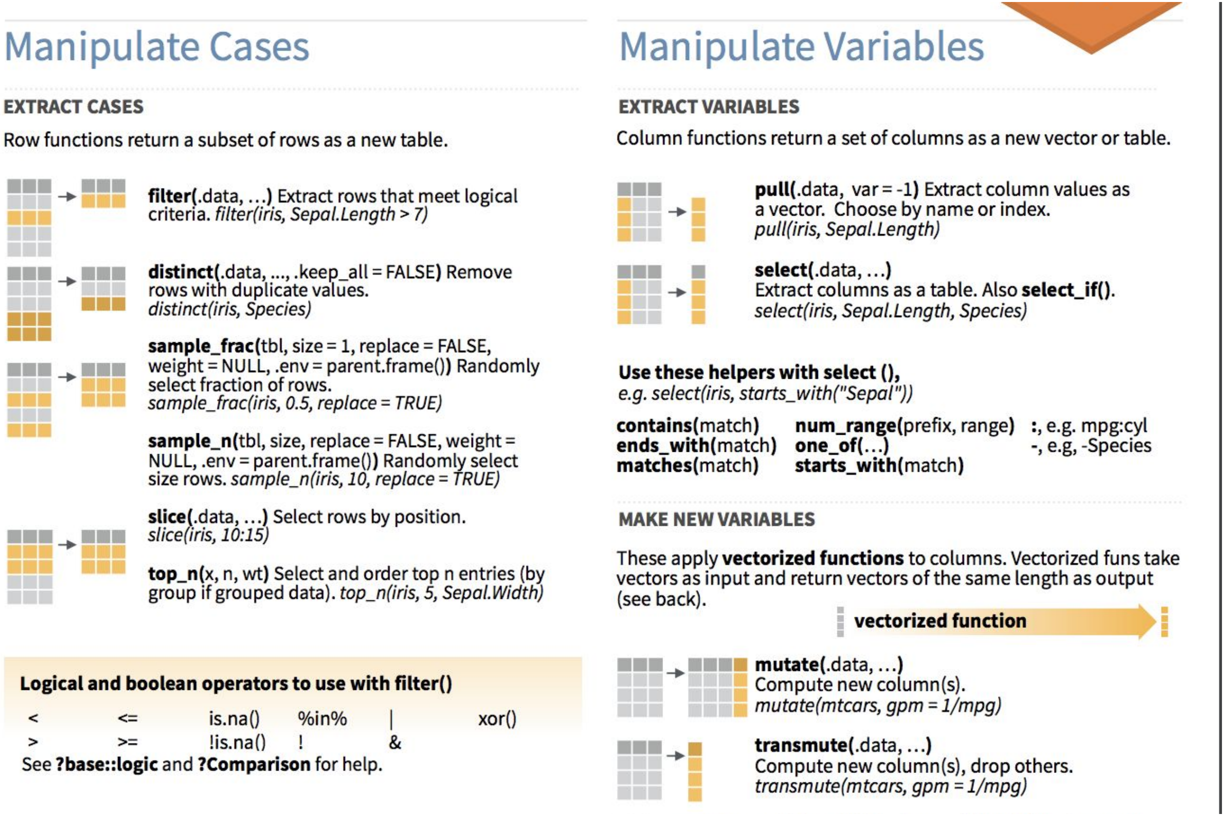
**add_column**(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

**rename**(.data, …) Rename columns.
*rename(iris, Length = Sepal.Length)*

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

# dplyr sample functions



## Manipulate Cases

### EXTRACT CASES

Row functions return a subset of rows as a new table.

**filter(**.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

**distinct(**.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values. *distinct(iris, Species)*

**sample_frac(**tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

**sample_n(**tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

**slice(**.data, …**)** Select rows by position. *slice(iris, 10:15)*

**top_n(**x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

**Logical and boolean operators to use with filter()**

| | | | | | |
|---|---|---|---|---|---|
| < | <= | is.na() | %in% | \| | xor() |
| > | >= | !is.na() | ! | & | |

See **?base::logic** and **?Comparison** for help.

## Manipulate Variables

### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

**pull(**.data, var = -1**)** Extract column values as a vector. Choose by name or index. *pull(iris, Sepal.Length)*

**select(**.data, …**)** Extract columns as a table. Also **select_if()**. *select(iris, Sepal.Length, Species)*

**Use these helpers with select (),** *e.g. select(iris, starts_with("Sepal"))*

| | | |
|---|---|---|
| **contains(**match**)** | **num_range(**prefix, range**)** | **:**, e.g. mpg:cyl |
| **ends_with(**match**)** | **one_of(**…**)** | **-**, e.g, -Species |
| **matches(**match**)** | **starts_with(**match**)** | |

### MAKE NEW VARIABLES

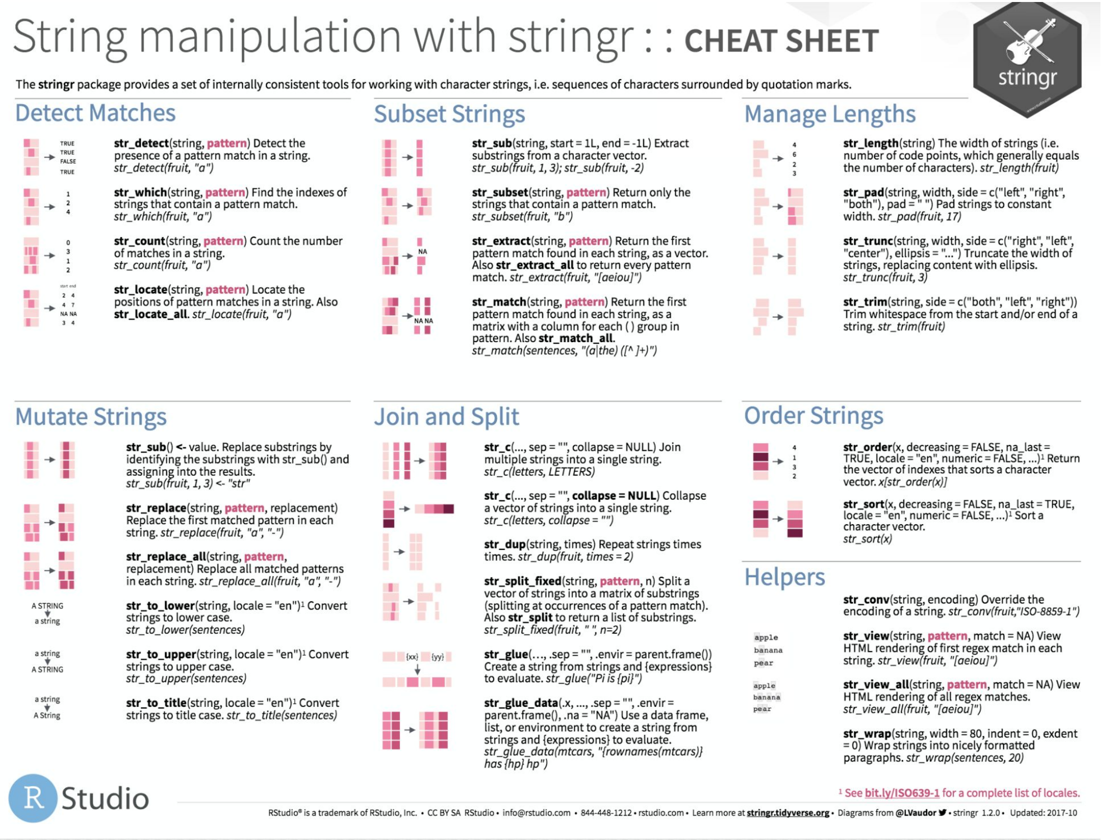
These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

**vectorized function**

**mutate(**.data, …**)** Compute new column(s). *mutate(mtcars, gpm = 1/mpg)*

**transmute(**.data, …**)** Compute new column(s), drop others. *transmute(mtcars, gpm = 1/mpg)*

# **stringr** package helps with string manipulation



stringr
TRUE
TRUE
FALSE
TRUE
1
2
4
0
3
1
2
start end
2 4
4 7
NA NA
3 4
NA
NA NA
4
6
2
3
A STRING
a string
a string
A STRING
a string
A String
{xx}
{yy}
4
1
3
2
apple
banana
pear
apple
banana
pear

## String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

### Detect Matches

**str_detect**(string, **pattern**) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*

**str_which**(string, **pattern**) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*

**str_count**(string, **pattern**) Count the number of matches in a string. *str_count(fruit, "a")*

**str_locate**(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

### Subset Strings

**str_sub**(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*

**str_subset**(string, **pattern**) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*

**str_extract**(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

**str_match**(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^ ]+)")*

### Manage Lengths

**str_length**(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

**str_pad**(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

**str_trunc**(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

**str_trim**(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

### Mutate Strings

**str_sub**() **<-** value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*

**str_replace**(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

**str_replace_all**(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

**str_to_lower**(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

**str_to_upper**(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

**str_to_title**(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

### Join and Split

**str_c**(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

**str_c**(..., sep = "", **collapse = NULL**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

**str_dup**(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

**str_split_fixed**(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

**str_glue**(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

**str_glue_data**(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

### Order Strings

**str_order**(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

**str_sort**(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

### Helpers

**str_conv**(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

**str_view**(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

**str_view_all**(string, **pattern**, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

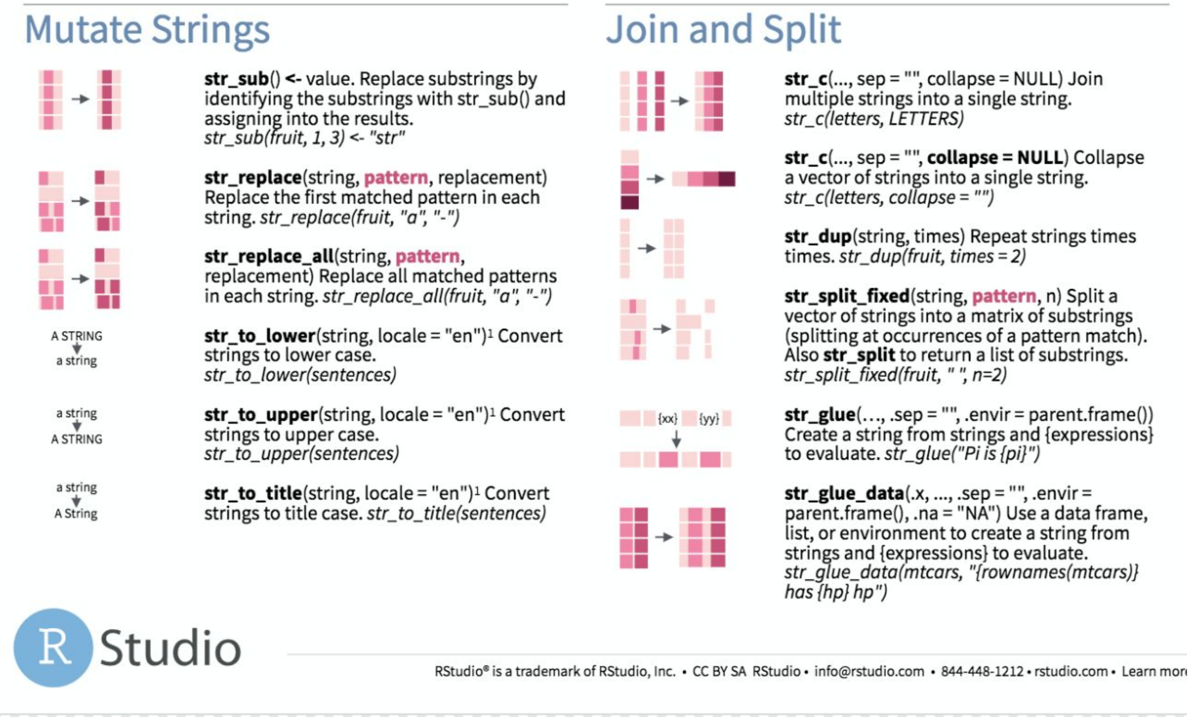
**str_wrap**(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **stringr.tidyverse.org** • Diagrams from **@LVaudor** • stringr 1.2.0 • Updated: 2017-10

# stringr sample functions



## Mutate Strings

**str_sub**() **<-** value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*

**str_replace**(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

**str_replace_all**(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

**str_to_lower**(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

**str_to_upper**(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

**str_to_title**(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

## Join and Split

**str_c**(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

**str_c**(..., sep = "", **collapse = NULL**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

**str_dup**(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

**str_split_fixed**(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

**str_glue**(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

**str_glue_data**(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn mor

# Helpful cheatsheets for data cleaning with R

Data wrangling: tidyr & dplyr

Data Transformation with dplyr (in-depth focus on dplyr functions)

String manipulation: stringr

All other R cheatsheets (from RStudio)